# Chapter 12. Loading and Manipulating Data

A lot of data is managed in and never leaves an Excel worksheet. But much of the data that Excel users work with comes from external databases such as SQL Server, Oracle, or Microsoft Access. You can work with data in various ways in Excel, usually by importing an entire table of data from a database or by using a query to import data that meets specific criteria. From a developer's perspective, you also have great programmatic control over the data you expose to users.

The Excel object model lets you create and manipulate queries from a variety of sources using the `QueryTable` object. If you want more programmatic control over your data, you have a choice of two programming interfaces. The ActiveX Data Objects (ADO) interface gives you access to data from a variety of data sources. The Data Access Objects (DAO) interface, which is native to Access databases, provides an easy-to-use interface for working with Access data.

In this chapter, I show how to:

- Work with `QueryTable` objects
- Work with `Parameter` objects
- Work with the ADO and DAO database programming interfaces

This chapter contains reference information for the following objects and their related collections: `QueryTable`, `Parameter`, `ADO.Command`, `ADO.Connection`, `ADO.Field`, `ADO.Parameter`, `ADO.Record`, `ADO.RecordSet`, `DAO.Database`, `DAO.DbEngine`, `DAO.Document`, `DAO.QueryDef`, and `DAO.Recordset`.

Code used in this chapter and additional samples are available in *ch12.xls*.



## 12.1. Working with QueryTable Objects

The `QueryTable` object gives you programmatic access to the database queries that are native to Excel. These database queries let you retrieve data from a variety of data sources and insert the data into your worksheets. In the Excel interface, you create a database query by clicking Import External Data, New Database Query on the Data menu.

In code, you create a database query by adding a `QueryTable` object to the `QueryTables` collection. When you do this, you supply a connection string to your data source as well as a destination on your worksheet where you want the results of the query to be inserted. For example, the following code inserts information for a specific product from the Products table of the Northwind Traders sample

database into the current worksheet, starting with the first cell of the worksheet:

```
Dim strConn As String
Dim strSQL As String
Dim qt As QueryTable

strConn = "ODBC;DSN=MS Access Database;" & _
    "DBQ=C:\Program Files\Microsoft Office\OFFICE11\SAMPLES\Northwind.mdb;"

Set qt = ActiveSheet.QueryTables.Add(Connection:=strConn, _
    Destination:=ActiveSheet.Range("A1"))
qt.CommandText = "SELECT * FROM Products WHERE (Products.ProductID=10)"
qt.Refresh
```

You can also use the ADO or DAO programming interfaces to create a recordset, and use the resulting `Recordset` object as your data source. To use either of these programming interfaces in Excel, you need to add a reference to the appropriate object library. On the Tools menu in the VBA programming environment, select References, then select the appropriate object library from the list. For example, the following code creates a query table using the Employees table in the Northwind Traders sample database and inserts the recordset name and data in the active worksheet:

```
Dim strDbPath As String
Dim db As DAO.Database
Dim rs As DAO.Recordset
Dim qt As QueryTable

strDbPath = "C:\Program Files\Microsoft Office\" & _
    "OFFICE11\SAMPLES\Northwind.mdb"

Set db = OpenDatabase(strDbPath)
Set rs = db.OpenRecordset("Employees")

Set qt = ActiveSheet.QueryTables.Add(Connection:=rs, _
    Destination:=ActiveSheet.Range("A3"))

ActiveSheet.Range("A1") = qt.Recordset.Name & " table:"

qt.Refresh
```

The ADO and DAO programming interfaces are discussed later in this chapter.

PREV PREV NEXT NEXT

## 12.2. QueryTable and QueryTables Members

Use the `QueryTables` collection to create new query tables and add them to a worksheet. Use the `Worksheet` object's `QuertyTables` property to get a reference to this collection. Use the `QueryTable` object to refresh the data in the query table and to control other aspects of the query. The `QueryTables` and `QueryTable` objects have the following members . Key members (shown in bold) are covered in the following reference section:

Web query members are covered in Chapter 24.

| **Add**[1] | **AdjustColumnWidth** |
|---|---|
| AfterRefresh | Application[2] |
| **BackgroundQuery** | BeforeRefresh |
| **CancelRefresh** | **CommandText** |
| **CommandType** | **Connection** |
| Count[1] | Creator[2] |
| **Delete** | **Destination** |
| EditWebPage | **EnableEditing** |
| **EnableRefresh** | **FetchedRowOverflow** |
| **FieldNames** | **FillAdjacentFormulas** |
| Item[1] | ListObject |
| **MaintainConnection** | Name |
| **Parameters** | Parent[2] |
| PostText | **PreserveColumnInfo** |
| **PreserveFormatting** | **QueryType** |
| **Recordset** | **Refresh** |
| **Refreshing** | **RefreshOnFileOpen** |
| **RefreshPeriod** | **RefreshStyle** |
| **ResetTimer** | **ResultRange** |
| RobustConnect | **RowNumbers** |
| SaveAsODC | SaveData |
| **SavePassword** | SourceConnectionFile |
| SourceDataFile | **TextFileColumnDataTypes** |
| **TextFileCommaDelimiter** | **TextFileConsecutiveDelimiter** |
| **TextFileDecimalSeparator** | **TextFileFixedColumnWidths** |
| **TextFileOtherDelimiter** | **TextFileParseType** |
| **TextFilePlatform** | **TextFilePromptOnRefresh** |
| **TextFileSemicolonDelimiter** | **TextFileSpaceDelimiter** |
| TextFileStartRow | **TextFileTabDelimiter** |

| | |
|---|---|
| **TextFileTextQualifier** | **TextFileThousandsSeparator** |
| **TextFileTrailingMinusNumbers** | **TextFileVisualLayout** |
| WebConsecutiveDelimitersAsOne | WebDisableDateRecognition |
| WebDisableRedirections | WebFormatting |
| WebPreFormattedTextToColumns | WebSelectionType |
| WebSingleBlockTextImport | WebTables |
| [1] Collection only | |
| [2] Object and collection | |

***querytables.Add(Connection, Destination, [Sql])***

Creates a new query table and adds it to the worksheet. Returns a `QueryTable` object.

| Argument | Description |
|---|---|
| *`Connection`* | A string or object reference identifying the source of the data. |
| *`Destination`* | A `Range` object identifying the upper-lefthand corner of the destination of the query table. |
| *`Sql`* | If the *`Connection`* argument is an ODBC data source, this argument is a string containing the SQL query to perform. Otherwise, including this argument either causes an error or is ignored. |

***querytable.AdjustColumnWidth [= setting]***

Set this property to False to disable the automatic adjustment for the best fit for columns in the specified query table.

***querytable.BackgroundQuery[= setting]***

True refreshes data in the query table asynchronously. False refreshes data synchronously. Default is True.

The `BeforeRefresh` and `AfterRefresh` events occur whether or not the query is refreshed synchronously or asynchronously. When synchronous, both events occur before the `Refresh` method completes. When asynchronous, only the `BeforeRefresh` event occurs before the `Refresh` method completes, then program flow continues.

### *querytable.CancelRefresh*

Cancels an asynchronous query. You can't refresh or delete a query while that query has refresh pending. When working with asynchronous queries, you should check the query table's `Refreshing` property and (possibly) cancel the pending refresh before deleting or refreshing that query again.

The following code cancels any pending refreshes before refreshing a query:

```
If qt.Refreshing Then qt.CancelRefresh
qt.Refresh
```

### *querytable.CommandText[= setting]*

Sets or returns the command string for the specified query table. The following code returns the results of a query with the specified command string:

```
Dim strConn As String
Dim strSQL As String
Dim qt As QueryTable

strConn = "ODBC;DSN=MS Access Database;" & _
    "DBQ=C:\Program Files\Microsoft Office\OFFICE11\SAMPLES\Northwind.mdb;"
Set qt = ActiveSheet.QueryTables.Add(Connection:=strConn, _
    Destination:=ActiveSheet.Range("A1"))

qt.CommandText = "SELECT * FROM Products WHERE (Products.ProductID=10)"
qt.Refresh
```

### *querytable.CommandType[= setting]*

Sets or returns the type of command string used by the specified query table. The type can be `xlCmdSQL`, a SQL string (default); `xlCmdCub`, a cube name for an online analytical processing (OLAP) data source; `xlCmdDefault`, command text that the OLE DB provider understands; or `xlCmdTable`, a table name for accessing OLE DB data sources.

### *querytable.Connection[= setting]*

Sets or returns the connection string for the specified query table. The following code creates a query table, returns its results, and displays the connection string in cell A6:

```
Dim strConn As String
Dim strSQL As String
```

```
Dim qt As QueryTable

strConn = "ODBC;DSN=MS Access Database;" & _
    "DBQ=C:\Program Files\Microsoft Office\OFFICE11\SAMPLES\Northwind.mdb;"

Set qt = ActiveSheet.QueryTables.Add(Connection:=strConn, _
    Destination:=ActiveSheet.Range("A1"))
qt.CommandText = "SELECT * FROM Products WHERE (Products.ProductID=10)"
qt.Refresh
ActiveSheet.Range("A6") = qt.Connection
```

### *querytable.Delete*

Deletes a query table. If the query table is refreshing asynchronously, `Delete` causes an error. Deleting a query table does not remove data from cells on a worksheetit just removes the ability to refresh those cells from their data source.

The following code deletes all of the query tables on the active worksheet and clears their data:

```
Dim qt As QueryTable
For Each qt In ActiveSheet.QueryTables
    If qt.Refreshing Then qt.CancelRefresh
    qt.Delete
Next
ActiveSheet.UsedRange.Clear
```

### *querytable.Destination*

Returns a `Range` object containing the cell in the upper-lefthand corner of the query table.

The following code selects the first cell of a query table on the active worksheet and asks if the user wants to delete it:

```
For Each qt In ActiveSheet.QueryTables
    qt.Destination.Select
    If MsgBox("Delete query table?", vbYesNo) = vbYes Then
        If qt.Refreshing Then qt.CancelRefresh
        qt.ResultRange.Clear
        qt.Delete
    End If
Next
```

### *querytable.EnableEditing[= setting]*

True allows the user to change the query definition through the Data menu's Import External Data

submenu. False disables the Import External Data menu items. Default is True.

***querytable.EnableRefresh[= setting]***

---

True allows the user to refresh the query through the Data menu's Refresh Data item. False disables the Refresh Data menu item. Default is True.

***querytable.FetchedRowOverflow[= setting]***

---

True if the number of rows returned by the last refresh of the specified query table is greater than the available number of rows.

***querytable.FieldNames[= setting]***

---

True if field names from the data source are displayed as column headings for the returned data. The following code specifies that field names will not be displayed in the query table:

```
ActiveSheet.QueryTables(1).FieldNames = False
```

***querytable.FillAdjacentFormulas[= setting]***

---

True causes calculated cells to the right of the query table to be repeated for each row when the query table is refreshed. False does not repeat adjacent formulas. Default is False.

Set `FillAdjacentFormulas` to True in order to create row totals, or other calculations, for each row in the query table automatically. To use this feature, create a query table, add a formula for the first row in the query table, set `FillAdjacentFormulas` to True, then refresh the data. For more information, see Chapter 24.

***querytable.MaintainConnection[= setting]***

---

This property returns True if the connection to the specified query table's data source is maintained after a refresh operation. You can set this property for queries to OLEDB sources only.

***querytable.Parameters***

---

Returns a `Parameters` collection object that represents the parameters of the specified query table.

Working with `Parameter` objects is covered later in this chapter.

***querytable.PreserveColumnInfo[= setting]***

---

True preserves the column sorting, filtering, and layout information when the specified query table is refreshed. False does not preserve formatting. Default is False.

***querytable.PreserveFormatting[= setting]***

---

True preserves the cell formatting of the query table when data is refreshed. False does not preserve formatting. Default is False.

If `PreserveFormatting` is True and a refresh imports new rows of data, formatting common to the first five rows of the query table is automatically applied to the new rows.

***querytable.QueryType[= setting]***

---

Returns a value identifying the type of data source used by the query table. Possible values are:

| |
|---|
| `xlTextImport` |
| `xlOLEDBQuery` |
| `xlWebQuery` |
| `xlADORecordset` |
| `xlDAORecordSet` |
| `xlODBCQuery` |

***querytable.Recordset[= setting]***

---

Sets or returns a `Recordset` object that serves as the data source for the specified query table. The following code creates a query table using the Employees table in the Northwind Traders sample database as the recordset and inserts the name of the recordset as well as the recordset data in the active worksheet:

```
Dim strDbPath As String
Dim db As DAO.Database
Dim rs As DAO.Recordset
Dim qt As QueryTable

strDbPath = "C:\Program Files\Microsoft Office\" & _
```

```
    "OFFICE11\SAMPLES\Northwind.mdb"

Set db = OpenDatabase(strDbPath)
Set rs = db.OpenRecordset("Employees")

Set qt = ActiveSheet.QueryTables.Add(Connection:=rs, _
    Destination:=ActiveSheet.Range("A3"))

ActiveSheet.Range("A1") = qt.Recordset.Name & " table:"

qt.Refresh
```

### *querytable.Refresh([BackgroundQuery])*

Refreshes a query table from its data source. Returns True if the refresh was submitted successfully, False if the user canceled the refresh.

| Argument | Description |
|---|---|
| *BackgroundQuery* | True refreshes the data asynchronously; False refreshes the data synchronously. Default is True. |

Most types of query table store connection and data source information that is used by `Refresh`. The exception is recordset queriesyou must set a new recordset before calling `Refresh` for query tables based on recordsets. See the `Recordset` property for an example.

When refreshing asynchronously, check the `Refreshing` property before calling `Refresh`. Otherwise, pending refreshes will cause an error. The following code cancels any pending asynchronous refresh before refreshing a query table:

```
If qt.Refreshing Then qt.CancelRefresh
qt.Refresh
```

### *querytable.Refreshing*

Returns True if an asynchronous refresh is pending for this query table, False if no refresh is pending.

### *querytable.RefreshOnFileOpen[= setting]*

True refreshes the query table when the workbook is opened; False does not refresh on open. Default is False.

### *querytable.RefreshPeriod[= setting]*

Sets or returns the number of minutes between automatic refreshes. The default is 0, for no automatic refreshing. You can set automatic refreshing on synchronous or asynchronous queries. `RefreshPeriod` is ignored for query tables created from recordsets.

The following code creates a query table from an ODBC data source and sets the query table to refresh once a minute:

```
Dim strConn As String
Dim strSQL As String
Dim qt As QueryTable

strConn="ODBC;DRIVER=SQL Server;SERVER=.;UID=Jeff;APP=Microsoft Office "& _
"XP;WSID=WOMBAT2;DATABASE=pubs;Trusted_Connection=Yes"
strSQL = "SELECT titles.title, titles.price, titles.pubdate, titles.ytd_sales
FROM pubs.dbo.titles titles"
Set qt = ActiveSheet.QueryTables.Add(strConn, [QueryDestination], strSQL)
qt.RefreshPeriod = 1
qt.Refresh
```

### *querytable.RefreshStyle[= setting]*

Determines how the query affects surrounding items on the worksheet when the query table is refreshed.

| Setting | Description |
|---|---|
| xlInsertDeleteCells | Inserts or deletes new rows and columns created by the query, moving surrounding items up or down and to the right or left as needed (default). |
| xlOverwriteCells | No new rows or columns are added to the worksheet. Surrounding items are overwritten as needed. |
| xlInsertEntireRows | Inserts a new row for each record returned by the query. Shifts existing items down as needed to accommodate the number of records returned. |

The following code modifies an existing query table to insert new rows on the worksheet as needed, shifting existing items on the worksheet down:

```
Set qt = ActiveSheet.QueryTables(1)
qt.RefreshStyle = xlInsertEntireRows
qt.Refresh
```

If a subsequent query reduces the number of records returned, the contents of the query table are replaced, but the rows that were previously shifted down are not shifted back up again as they would be if `RefreshStyle` was set to `xlInsertDeleteCells`.

### *querytable.ResetTimer*

Resets the timer used for periodic queries, in effect delaying when a query occurs. Use the `RefreshPeriod` property to automatically refresh a query periodically.

### *querytable.ResultRange*

Returns the range containing the results of the query. For example, the following code clears the results from a query table on the active worksheet:

```
ActiveSheet.QueryTables(1).ResultRange.Clear
```

If a query table has been created but not yet refreshed, accessing `ResultRange` causes an error. There's no direct way to test whether a query table has been refreshed. One solution to this problem is to write a helper function similar to the following to check if a query table has a result before accessing `ResultRange` elsewhere in code:

```
Public Function HasResult(qt As QueryTable) As Boolean
    Dim ret As Boolean
    On Error Resume Next
    Debug.Print qt.ResultRange.Address
    If Err Then ret = False Else ret = True
    On Error GoTo 0
    HasResult = ret
End Function
```

Now, you can easily test if a query table has a result before clearing the result range or performing other tasks as shown here:

```
Set qt = ActiveSheet.QueryTables(1)
If HasResult(qt) Then qt.ResultRange.Clear
```

### *querytable.RowNumbers[= setting]*

Set this property to True to display row numbers in the first column of the specified query table. The numbers do not display until the query table is refreshed. They will be reset each time the query table is refreshed. The following code adds row numbers to the first query table on the active worksheet:

```
With ActiveSheet.QueryTables(1)
    .RowNumbers = True
    .Refresh
End With
```

***querytable.SavePassword[= setting]***

---

Set this property to True to save password information in an ODBC connection string with the specified query table.

***querytable.TextFileColumnDataTypes[= setting]***

---

Sets or returns an array of constants specifying the data types applied to a text file being imported into the specified query table.

***querytable.TextFileCommaDelimiter[= setting]***

---

True if you are using a comma delimiter when you are importing a text file into the specified query table.

***querytable.TextFileConsecutiveDelimiter[= setting]***

---

True if consecutive delimiters are treated as a single delimiter when you are importing a text file into the specified query table.

***querytable.TextFileDecimalSeparator[= setting]***

---

Sets or returns the decimal separator used when you are importing a text file into the specified query table.

***querytable.TextFileFixedColumnWidths[= setting]***

---

Sets or returns an array of integers that correspond to the widths of the columns in the text file that you are importing into the specified query table. The following code imports text from a sample file and places the characters in each row of the active worksheet as follows:

- The first five characters are placed in the first column.
- The next four characters are placed in the second column.
- The remaining characters are placed in the third column:

```
Dim strCnn As String
Dim qt As QueryTable

strCnn = "TEXT;C:\My Documents\qtsample.txt"
Set qt = ActiveSheet.QueryTables.Add(Connection:=strCnn,& _
    Destination:=ActiveSheet.Range("A1"))
With qt
    .TextFileParseType = xlFixedWidth
    .TextFileFixedColumnWidths = Array(5, 4)
    .Refresh
End With
```

***querytable.TextFileOtherDelimiter[= setting]***

---

Sets or returns the character used as a delimiter when you are importing a text file into the specified query table.

***querytable.TextFileParseType[= setting]***

---

Set this property to `xlFixedWidth` if the column data in the text file you are importing into the specified query table has a fixed width. Set this property to `xlDelimited` (default) if the column data in the text file is separated by a delimiter character.

***querytable.TextFilePlatform[= setting]***

---

Set this property to `xlMacintosh` if the text file you are importing into the specified query table originated on the Macintosh operating system. Set this property to `xlMSDOS` if the text file originated on the MS-DOS operating system. Set this property to `xlWindows` if the text file originated on the Windows operating system.

***querytable.TextFilePromptOnRefresh[= setting]***

---

True if you want to be prompted for the name of the text being imported into the specified query table each time the query table is refreshed.

***querytable.TextFileSemicolonDelimiter[= setting]***

---

True if you are using a semicolon delimiter when you are importing a text file into the specified query table.

***querytable.TextFileSpaceDelimiter[= setting]***

---

True if you are using a space character delimiter when you are importing a text file into the specified query table.

***querytable.TextFileTabDelimiter[= setting]***

---

True if you are using a tab character delimiter when you are importing a text file into the specified query table.

***querytable.TextFileTextQualifier[= setting]***

---

Set this property to `xlTextQualifierSingleQuote` if the text file you are importing into the specified query table uses single quotes rather than double quotes to indicate what is enclosed between the quotes is text. Set this property to `xlTextQualifierNone` if the file does not use quotes to indicate a text string. Set this property to `xlTextQualifierDoubleQuote` (default) if the file uses double quotes as a text qualifier.

***querytable.TextFileThousandsSeparator[= setting]***

---

Sets or returns the thousands separator used when you are importing a text file into the specified query table.

***querytable.TextFileTrailingMinusNumbers[= setting]***

---

True if numbers imported into the specified query table that begin with the hyphen character (-) are treated as negative numbers. False if they are treated as text.

***querytable.TextFileVisualLayout[= setting]***

---

Sets or returns the left-to-right layout of text for text imported into the specified query table. When the property is set to 1, layout is left-to-right. When the property is set to 2, the layout is right-to-left.

PREV PREV NEXT NEXT

## 12.3. Working with Parameter Objects

The `Parameter` object lets you supply parameter criteria to limit the data returned by a query. This is useful if you want to create a query that returns a general set of data but you want to work with different subsets of that data or different individual records. You can supply different parameters rather than creating a new query for each subset or record.

You create a parameter by adding a `Parameter` object to the `Parameters` collection of a `QueryTable` object. You can then supply a specific parameter value or use a value in a cell on your worksheet. For example, the following code creates a query table that uses the value in cell A1 as the parameter:

```
Dim strConn As String
Dim strSQL As String
Dim qt As QueryTable
Dim param As Parameter

strConn = "ODBC;DSN=MS Access Database;" & _
    "DBQ=C:\Program Files\Microsoft Office\OFFICE11\SAMPLES\Northwind.mdb;"

Set qt = ActiveSheet.QueryTables.Add(Connection:=strConn, _
    Destination:=ActiveSheet.Range("C1"))
qt.CommandText = "SELECT * FROM Products WHERE (Products.ProductID=?)"
Set param = qt.Parameters.Add("ProductsParam")
param.SetParam xlRange, Range("A1")
qt.Refresh
```

PREV PREV NEXT NEXT

## 12.4. Parameter Members

Use the `Parameters` collection to add parameters to the SQL query used by a query table. Use the `QueryTable` object's `Parameters` property to get a reference to this collection. Use the `Parameter` object to set the contents of the parameter. The `Parameters` collection and `Parameter` object have the following members . Key members (shown in bold) are covered in the following reference section:

| | |
|---|---|
| **`Add`**[1] | Application[2] |
| Count[1] | Creator[2] |
| **`DataType`** | **`Delete`** |
| Item[1] | Name |
| Parent[2] | **`PromptString`** |
| **`RefreshOnChange`** | **`SetParam`** |
| **`SourceRange`** | **`Type`** |
| **`Value`** | |
| [1] Collection only | |
| [2] Object and collection | |

## *parameters.Add(Name, [iDataType])*

Creates a new query parameter. Returns a `Parameter` object.

| Argument | Description |
|---|---|
| *Name* | A string that identifies the parameter. |
| *iDataType* | If you want to specify a data type for the parameter, use one of the following constants:<br><br>xlParamTypeBigInt<br>xlParamTypeBinary<br>xlParamTypeBit<br>xlParamTypeChar<br>xlParamTypeDate<br>xlParamTypeDecimal<br>xlParamTypeDouble<br>xlParamTypeFloat<br>xlParamTypeInteger<br>xlParamTypeLongVarBinary<br>xlParamTypeWChar<br>xlParamTypeNumeric<br>xlParamTypeLongVarChar<br>xlParamTypeReal<br>xlParamTypeSmallInt<br>xlParamTypeTime<br>xlParamTypeTimeStamp<br>xlParamTypeTinyInt<br>xlParamTypeUnknown<br>xlParamTypeVarBinary<br>xlParamTypeVarChar |

The following code creates a query table that uses a parameter to supply the product ID to the underlying query. The `?` character is a placeholder for the query value, which in this case is the value `10` for the `ProductID`:

```
Dim strConn As String
Dim qt As QueryTable
Dim param As Parameter

strConn = "ODBC;DSN=MS Access Database;" & _
    "DBQ=C:\Program Files\Microsoft Office\OFFICE11\SAMPLES\Northwind.mdb;"

Set qt = ActiveSheet.QueryTables.Add(Connection:=strConn, _
    Destination:=ActiveSheet.Range("A1"))
qt.CommandText = "SELECT * FROM Products WHERE (Products.ProductID=?)"
Set param = qt.Parameters.Add("ProductsParam")
param.SetParam xlConstant, 10
qt.Refresh
```

***parameter.DataType[= setting]***

Sets or returns the data type of the specified parameter. See the `Add` method for a list of possible values.

***parameter.Delete***

Deletes the specified parameter.

***parameter.PromptString[= setting]***

If the specified parameter uses a prompt string, this property returns the prompt string. The following code creates two parameter query tables on the active worksheet and uses the same prompt string for both:

```
Dim strConn As String
Dim qt As QueryTable
Dim param As Parameter

strConn = "ODBC;DSN=MS Access Database;" & _
    "DBQ=C:\Program Files\Microsoft Office\OFFICE11\SAMPLES\Northwind.mdb;"

Set qt1 = ActiveSheet.QueryTables.Add(Connection:=strConn, _
    Destination:=ActiveSheet.Range("A1"))
qt1.CommandText = "SELECT * FROM Products WHERE (Products.ProductID=?)"
Set param1 = qt1.Parameters.Add("ProductsParam1")
param1.SetParam xlPrompt, "Please enter a Product ID."

Set qt2 = ActiveSheet.QueryTables.Add(Connection:=strConn, _
Destination:=ActiveSheet.Range("A5"))
qt2.CommandText = "SELECT * FROM Products WHERE (Products.ProductID=?)"
Set param2 = qt2.Parameters.Add("ProductsParam2")
param2.SetParam xlPrompt, param1.PromptString




qt1.Refresh
qt2.Refresh
```

***parameter.RefreshOnChange[= setting]***

If the specified parameter uses a single-cell range as a parameter value, this property refreshes the query

table whenever the cell value changes.

***parameter.SetParam[= setting]***

---

Defines the specified parameter. The following code creates a query table that uses the value in cell A1 as the parameter:

```
Dim strConn As String
Dim strSQL As String
Dim qt As QueryTable
Dim param As Parameter

strConn = "ODBC;DSN=MS Access Database;" & _
    "DBQ=C:\Program Files\Microsoft Office\OFFICE11\SAMPLES\Northwind.mdb;"

Set qt = ActiveSheet.QueryTables.Add(Connection:=strConn, _
    Destination:=ActiveSheet.Range("C1"))
qt.CommandText = "SELECT * FROM Products WHERE (Products.ProductID=?)"
Set param = qt.Parameters.Add("ProductsParam")
param.SetParam xlRange, Range("A1")
qt.Refresh
```

***parameter.SourceRange[= setting]***

---

If the specified parameter uses a single-cell range as its parameter, returns the corresponding `Range` object.

***parameter.Type[= setting]***

---

Sets or returns the type of the specified parameter, either `xlConstant` if the parameter is a constant, `xlPrompt` if it is a prompt string, or `xlRange` if it is a single-cell range.

***parameter.Value[= setting]***

---

Sets or returns the value of the specified parameter, either a constant, a prompt string, or a single-cell `Range` object.

PREV PREV NEXT NEXT

## 12.5. Working with ADO and DAO

If you want to be able to manage all the details of working with data in your worksheets, you can manipulate data programmatically using one of the two programming interfaces: ActiveX Data Objects (ADO) and Data Access Objects (DAO).

DAO came first. It was developed in conjunction with Microsoft Access and is the native programming interface for the Jet database engine, the built-in data engine for Access. ADO came later, incorporating some of the database cursor optimization that came with Microsoft's acquisition of FoxPro. It is more flexible, better suited for high-performance applications, and designed to be more neutral in dealing with different data sources. But, truth be told, many experienced and respected Access developers still do most of their work in DAO.

To use either of these programming interfaces in Excel, you need to add a reference to the appropriate object library. On the Tools menu in the VBA programming environment, select References, then select the appropriate object library from the list.

A full discussion of ADO and DAO is beyond the scope of this book, but we will touch on some of the key objects and members of each interface.

PREV PREV NEXT NEXT

## 12.6. ADO Objects and Members

The ADO object model includes the key objects listed in the following table. There are additional objects, but these cover the fundamentals of working with ADO. For information about the additional objects, see the ADO Help.

| Object | Description |
|---|---|
| Command | Defines a specific commandsuch as a SQL statement, table name, or stored procedurethat returns data from a data source. |
| Connection | Represents a connection to a data source. |
| Field | Represents a field of data from a data source. |
| Parameter | Represents a parameter associated with a specific command. |
| Record | Represents a single record in a recordset. |
| Recordset | Represents a set of records from a table or command. |

Descriptions of the members of these objects follow. Key members (shown in bold) are covered in the following reference sections.

### 12.6.1. ADO.Command Members

| | |
|---|---|
| **`ActiveConnection`** | Cancel |
| CommandStream | **`CommandText`** |

| | |
|---|---|
| CommandTimeout | **CommandType** |
| **CreateParameer** | Dialect |
| **Execute** | Name |
| Prepared | Properties |
| State | |

***command.ActiveConnection[= setting]***

Sets or returns the connection used by the specified command. The following code returns a record by executing a SQL command using the active connection:

```
Dim cnn As ADODB.Connection
Dim cmd As ADODB.Command
Dim rs As ADODB.Recordset
Dim strDbPath As String

Set cnn = New ADODB.Connection
Set cmd = New ADODB.Command
Set rs = New ADODB.Recordset
strDbPath = "C:\Program Files\Microsoft Office\OFFICE11\SAMPLES\Northwind.mdb"
cnn.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;" _
    & "Data Source=" & strDbPath
cnn.Open
Set cmd.ActiveConnection = cnn
cmd.CommandText = "SELECT * FROM Employees Where EmployeeID = 9;"
Set rs = cmd.Execute

Set qt = ActiveSheet.QueryTables.Add(Connection:=rs, _
    Destination:=ActiveSheet.Range("A3"))
qt.Refresh

ActiveSheet.Range("A1") = qt.Recordset.Source
rs.Close
cnn.Close
Set rs = Nothing
Set cnn = Nothing
```

***command.CommandText[= setting]***

Sets or returns the command text used by the specified command. See the `ActiveConnection` code example for an example of using `CommandText`.

***command.CommandType[= setting]***

Sets or returns the type of the specified command: `adCmdUnspecified`, `adCmdText`, `adCmdTable`, `adCmdStoredProc`, `adCmdUnknown`, `adCmdFile`, or `adCmdTableDirect`.

***command.CreateParameter***

---

Creates a new parameter for the specified command.

***command.Execute***

---

Executes the specified command. See the `ActiveConnection` code example for an example of using `Execute`.

***command.Name[= setting]***

---

Sets or returns the name of the specified command.

### 12.6.2. ADO.Connection Members

| Attributes | **`BeginTrans`** |
|---|---|
| **`Cancel`** | Close |
| **`CommandTimeout`** | **`CommitTrans`** |
| **`ConnectionString`** | **`ConnectionTimeout`** |
| CursorLocation | DefaultDatabase |
| Execute | IsolationLevel |
| Mode | **`Open`** |
| OpenSchema | Provider |
| **`RollbackTrans`** | State |
| **`Version`** | |

***connection.BeginTrans***

---

Begins a transactiona series of operations performed as a whole (committed) or canceled (rolled back). The following code wraps the code example used for the `Command` object's `ActiveConnection` method around a transaction so that it can be committed or rolled back:

```
Dim cnn As ADODB.Connection
Dim cmd As ADODB.Command
Dim rs As ADODB.Recordset
Dim strDbPath As String

Set cnn = New ADODB.Connection
Set cmd = New ADODB.Command
Set rs = New ADODB.Recordset
strDbPath = "C:\Program Files\Microsoft Office\OFFICE11\SAMPLES\Northwind.mdb"
cnn.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;" _
    & "Data Source=" & strDbPath
cnn.Open
Set cmd.ActiveConnection = cnn
cnn.BeginTrans

cmd.CommandText = "SELECT * FROM Employees Where EmployeeID = 9;"
Set rs = cmd.Execute

' Prompt user to commit all changes made
If MsgBox("Save all changes?", vbYesNo) = vbYes Then
    cnn.CommitTrans
    Set qt = ActiveSheet.QueryTables.Add(Connection:=rs, _
    Destination:=ActiveSheet.Range("A3"))
    qt.Refresh
    ActiveSheet.Range("A1") = qt.Recordset.Source
Else
    cnn.RollbackTrans
End If

rs.Close
cnn.Close
Set rs = Nothing
Set cnn = Nothing
```

### *connection.Cancel*

Cancels the specified connection object's last `Execute` or `Open` operation.

### *connection.CommandTimeout[= setting]*

Specifies the time to wait, in seconds, while executing a command on the specified connection before terminating it.

### *connection.CommitTrans*

Saves any changes made during a transaction. See the `BeginTrans` code example for an example of using `CommitTrans`.

***connection.ConnectionString[= setting]***

Specifies the connection string used to connect to a data source. See the `BeginTrans` code example for an example of using `ConnectionString`.

***connection.ConnectionTimeout[= setting]***

Specifies the time to wait, in seconds, while establishing a connection before terminating it.

***connection.Open***

Opens the specified connection. See the `BeginTrans` code example for an example of using `Open`.

***connection.RollbackTrans***

Cancels any changes made during a transaction. See the `BeginTrans` code example for an example of using `RollbackTrans`.

***connection.Version[= setting]***

Returns the ADO version number.

### 12.6.3. ADO.Field and ADO.Fields Members

| | |
|---|---|
| **`ActualSize`** | Append[1] |
| **`AppendChunk`** | Attributes |
| **`CancelUpdate`**[1] | Count[1] |
| **`DefinedSize`** | Delete[1] |
| **`GetChunk`** | Item[1] |
| Name | **`NumericScale`** |
| **`OriginalValue`** | Precision |
| Refresh[1] | Resync[1] |
| Status | Type |
| | |

| | |
|---|---|
| **`UnderlyingValue`** | Update[1] |
| **`Value`** | |
| [1] Collection only | |

### *field.ActualSize[= setting]*

Returns the actual size of the data in a field. Use the `DefinedSize` property to return the size of data that the field is capable of holding.

### *field.AppendChunk*

Appends data to a large text or binary field.

### *fields.CancelUpdate*

Cancels any updates made to the specified `Fields` collection.

### *field.DefinedSize[= setting]*

Returns the size of data that the field is capable of holding. Use the `ActualSize` property to return the actual size of the data in a field.

### *field.GetChunk(Size)*

Returns all or a specified portion of a large text or binary file.

| Argument | Description |
|---|---|
| *`Size`* | The number of bytes or characters that you want to return |

### *field.NumericScale[= setting]*

Sets or returns the number of decimal places to use for numeric values.

***field.OriginalValue[= setting]***

Returns the value of a field before any changes were made. Use this property with the `UnderlyingValue` property in a multiuser environment when you want to make sure that you are using the most current data.

***field.UnderlyingValue[= setting]***

Returns the current value of a field. Use this property with the `OriginalValue` property in a multiuser environment when you want to make sure that you are using the most current data.

***field.Value[= setting]***

Sets or returns the value of data stored in the specified field.

### 12.6.4. ADO.Parameter and ADO.Parameters Members

| | |
|---|---|
| Append[1] | **`AppendChunk`** |
| Attributes | Count[1] |
| Delete[1] | Direction |
| Item[1] | **`Name`** |
| **`NumericScale`** | **`Precision`** |
| Properties | Refresh[1] |
| **`Size`** | Type |
| **`Value`** | |
| [1] Collection only | |

***Parameter.AppendChunk***

Appends data to a large text or binary field.

***Parameter.Name[= setting]***

Sets or returns the name of the specified parameter.

***Parameter.NumericScale[= setting]***

---

Sets or returns the number of numeric decimal places in the specified parameter.

***Parameter.Precision[= setting]***

---

Sets or returns the maximum number of digits in a numeric parameter value.

***parameter.Size[= setting]***

---

Sets or returns the maximum size of the specified parameter, in bytes or characters.

***parameter.Value[= setting]***

---

Sets or returns the parameter's value.

### 12.6.5. ADO.Record Members

| | |
|---|---|
| **`ActiveConnection`** | **`Cancel`** |
| Close | CopyRecord |
| DeleteRecord | **`GetChildren`** |
| Mode | MoveRecord |
| **`Open`** | ParentURL |
| Properties | **`RecordType`** |
| **`Source`** | **`State`** |

***record.ActiveConnection[= setting]***

---

Sets or returns the connection used by the specified record.

***record.Cancel***

Cancels a pending `CopyRecord`, `DeleteRecord`, `MoveRecord`, or `Open operation`.

### *record.GetChildren*

Returns a `Recordset` object whose rows are children of the specified record in a parent-child relationship.

### *record.Open([Source], [ActiveConnection], [Mode]), [CreateOptions], [Options], [UserName], [Password])*

Opens the record.

| Argument | Description |
|---|---|
| *`Source`* | If the record source has not already been specified, you can specify a `Command`, `Record`, or `Recordset` object; table; or SQL statement as the source. |
| *`ActiveConnection`* | If the connection has not already been specified, you can specify a `Connection` object or connect stiring. |
| *`Mode`* | If the mode has not already been specified, you can specify a `ConnectModeEnum` constant value that specifies the access mode. The value can be `adModeRead`, `adModeReadWrite`, `adModeRecursive`, `adModeShareDenyNone`, `adModeShareDenyRead`, `adModeShareDenyWrite`, `adModeShareExclusive`, `adModeUnknown`, `adModeWrite`. |
| *`CreateOptions`* | Lets you specify whether an existing file or directory should be opened or a new file or directory should be created. |
| *`Options`* | Lets you specify options for opening the record. The value can be `adDelayFetchFields`, `adDelayFetchStream`, `adOpenAsync`, `adOpenExecuteCommand`, `adOpenRecordUnspecified`, or `adOpenOutput`. |
| *`UserName`* | Lets you specify a username granting access to *`Source`*. |
| *`Password`* | Lets you specify a password for the username. |

### *record.RecordType*

Returns the `Record` object type, either `adSimpleRecord`, `adCollectionRecord`, `adRecordUnknown`, or `adStructDoc`.

***record.Source[= setting]***

Sets or returns the data source for the record.

***record.State***

Returns the state of the record, either `adStateClosed`, `adStateOpen`, `adStateConnecting`, `adStateExecuting`, or `adStateFetching`.

### 12.6.6. ADO.Recordset Members

| | |
|---|---|
| AbsolutePage | **`AbsolutePosition`** |
| **`ActiveCommand`** | **`ActiveConnection`** |
| **`AddNew`** | **`BOF`** |
| Bookmark | CacheSize |
| **`Cancel`** | CancelBatch |
| **`CancelUpdate`** | Clone |
| Close | CompareBookmarks |
| CursorLocation | CursorType |
| DataMember | DataSource |
| **`Delete`** | EditMode |
| **`EOF`** | **`Filter`** |
| Find | GetRows |
| GetString | Index |
| LockType | MarshalOptions |
| MaxRecords | Move |
| **`MoveFirst`** | **`MoveLast`** |
| **`MoveNext`** | **`MovePrevious`** |
| NextRecordset | **`Open`** |
| PageCount | PageSize |
| **`RecordCount`** | **`Requery`** |
| Resync | Save |
| Seek | Sort |
| **`Source`** | State |

| | |
|---|---|
| Status | StayInSync |
| Supports | **Update** |
| UpdateBatch | |

***recordset.AbsolutePosition[= setting]***

Sets or returns the ordinal position of the current record in the recordset.

***recordset.ActiveCommand[= setting]***

Returns the `Command` object used to create the recordset.

***recordset.ActiveConnection [= setting]***

Sets or returns the connection string or `Connection` object used by the recordset.

***recordset.AddNew([FieldList], [Values])***

Creates a new record.

| Argument | Description |
|---|---|
| *`FieldList`* | A single field name or an array of names or ordinal numbers specifying the fields in the new record |
| *`Values`* | A single field value or an array of values for the fields |

The following code adds a new record to the Employees table in the Northwind Traders sample database using cell values on the current worksheet:

```
Dim cnn As ADODB.Connection
Dim cmd As ADODB.Command
Dim rs As ADODB.Recordset
Dim strDbPath As String
Dim strConnect As String

Set cnn = New ADODB.Connection
Set cmd = New ADODB.Command
Set rs = New ADODB.Recordset
strDbPath = "C:\Program Files\Microsoft Office\OFFICE11\SAMPLES\Northwind.mdb"
```

```
cnn.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;" _
    & "Data Source=" & strDbPath

cnn.Open
rs.Open "Employees", cnn, adOpenDynamic, adLockOptimistic, adCmdTable

rs.AddNew
rs!LastName = ActiveSheet.Range("B4")
rs!FirstName = ActiveSheet.Range("C4")
rs.Update

rs.Close
cnn.Close
Set rs = Nothing
Set cnn = Nothing
```

### *recordset.BOF[= setting]*

True if the current record position is before the first record in the recordset.

### *recordset.Cancel*

Cancels the last `Open` operation for the recordset.

### *recordset.CancelUpdate*

Cancels any pending changes for the current record.

### *recordset.Delete([AffectRecords])*

Deletes the current record or a group of records.

| Argument | Description |
|---|---|
| *`AffectRecords`* | A constant that specifies the records affected by the delete operation, either `adAffectAll`, `adAffectAllChapters`, `adAffectCurrent`, or `adAffectGroup`. |

### *recordset.EOF[= setting]*

True if the current record position is after the last record in the recordset. The following code uses the

EOF property to test for the end of the recordset, adding names from the Employees table in the Northwind Traders sample database to the first column of the current worksheet:

```
Dim cnn As ADODB.Connection
Dim cmd As ADODB.Command
Dim rs As ADODB.Recordset
Dim strDbPath As String
Dim intIdx As Integer

Set cnn = New ADODB.Connection
Set cmd = New ADODB.Command
Set rs = New ADODB.Recordset
strDbPath = "C:\Program Files\Microsoft Office\OFFICE11\SAMPLES\Northwind.mdb"
cnn.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;" _
    & "Data Source=" & strDbPath

cnn.Open
rs.Open "Employees", cnn, adOpenStatic, adLockReadOnly, adCmdTable

rs.MoveFirst
intIdx = 1
Do Until rs.EOF
    strName = rs!FirstName & " " & rs!LastName
    ActiveSheet.Cells(intIdx, 1) = strName
    rs.MoveNext
    intIdx = intIdx + 1
Loop

rs.Close
cnn.Close
Set rs = Nothing
Set cnn = Nothing
```

### *recordset.Filter[= setting]*

---

Sets or returns a filter for the recordset. You can use filters to work with different sets of data in a table without having to open separate recordsets. The following code adds product names for all beverages from the Products table in the Northwind Traders sample database to the first column of the current worksheet:

```
Dim cnn As ADODB.Connection
Dim cmd As ADODB.Command
Dim rs As ADODB.Recordset
Dim strDbPath As String
Dim intIdx As Integer

Set cnn = New ADODB.Connection
Set cmd = New ADODB.Command
Set rs = New ADODB.Recordset
strDbPath = "C:\Program Files\Microsoft Office\OFFICE11\SAMPLES\Northwind.mdb"
cnn.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;" _
    & "Data Source=" & strDbPath

cnn.Open
```

```
rs.Open "Products", cnn, adOpenStatic, adLockReadOnly, adCmdTable
rs.Filter = "CategoryID = 1"

rs.MoveFirst
intIdx = 1
Do Until rs.EOF
    strName = rs!ProductName
    ActiveSheet.Cells(intIdx, 1) = strName
    rs.MoveNext
    intIdx = intIdx + 1
Loop

rs.Close
cnn.Close
Set rs = Nothing
Set cnn = Nothing
```

### *recordset.MoveFirst*

Moves to the first record in the recordset.

### *recordset.MoveLast*

Moves to the last record in the recordset.

### *recordset.MoveNext*

Moves to the next record in the recordset. See the `EOF` and `Filter` code examples for examples of using `MoveNext`.

### *recordset.MovePrevious*

Moves to the previous record in the recordset.

### *recordset.Open([Source], [ActiveConnection], [CursorType] , [LockType] , [Options])*

Opens the recordset for database operations.

| Argument | Description |
|---|---|
| | |

| | |
|---|---|
| *Source* | The source of the recordset. The source can be a `Command` object, an SQL statement, a table name, a stored procedure call, a URL, or the name of a file or `Stream` object. |
| *ActiveConnection* | A `Connection` object or connection string. |
| *CursorType* | The type of database cursor to use for the recordset. The cursor can be `adOpenDynamic`, `adOpenForwardOnly` (default), `adOpenKeyset`, `adOpenStatic`, or `adOpenUnspecified`. |
| *LockType* | The type of locking to use for the recordset. The cursor can be `adLockBatchOptimistic`, `adLockOptimistic`, `adLockPessimistic`, `adLockReadOnly`, or `adLockUnspecified`. |
| *Options* | A constant specifying how a command source should be interpreted or executed. |

***recordset.RecordCount[= setting]***

Returns the number of records in the recordset.

***recordset.Requery***

Updates the recordset by running the query on which it is based.

***recordset.Source[= setting]***

Returns a string or `Command` object indicating the source of the recordset.

***recordset.Update([Fields], [Value])***

Saves changes made to the current record.

| Argument | Description |
|---|---|
| *Fields* | The name of the field being updated or an array of names or ordinal positions if you are updating multiple fields |
| *Value* | The updated value of the field or an array of values if you are updating multiple fields |

See the `AddNew` code example for an example of using `Update`.

## 12.7. DAO Objects and Members

The DAO object model includes the key objects listed in the following table. There are additional objects, but these cover the fundamentals of working with DAO. For information about the additional objects, see the DAO Help.

| Object | Description |
|---|---|
| `Database/Databases` | The `Database` object represents an open database. The `Databases` collection contains all open databases. |
| DbEngine | Represents the Jet database engine. It is the top-level object in the DAO object model. |
| `Document/Documents` | The `Document` object represents information about an instance of a Microsoft Access object, such as a form or report. The `Documents` collection contains all the `Document` objects of the same type. |
| `QueryDef/QueryDefs` | The `QueryDef` object represents a Microsoft Access query. The `QueryDefs` collection contains all the queries in a database. |
| `Recordset/Recordsets` | The `Recordset` object represents a set of records from a table or query. The `Recordsets` collection contains all open recordsets in a database. |

Descriptions of the members of these objects follow. Key members (shown in bold) are covered in the following reference sections.



## 12.8. DAO.Database and DAO.Databases Members

| | |
|---|---|
| Close | CollatingOrder |
| Connect | **`Connection`** |
| Containers | Count[1] |
| CreateProperty | CreateQueryDef |
| CreateRelation | CreateTableDef |
| DesignMasterID | **`Execute`** |
| MakeReplica | Name |
| NewPassword | **`OpenRecordset`** |
| PopulatePartial | Properties |

| | |
|---|---|
| QueryTimeout | RecordsAffected |
| Refresh[1] | Relations |
| ReplicaID | Synchronize |
| Transactions | Updatable |
| Version | |

[1] Collection only

***database.Connection***

Returns the `Connection` object for the database.

***database.Execute(Source, [Options])***

Executes an action query or SQL statement.

| Argument | Description |
|---|---|
| *`Source`* | An SQL statement or the name of a query. |
| *`Options`* | A combination of constants that specify characteristics of the recordset. See DAO Help for more information about these options. |

***database.OpenRecordset(Source, [Type], [Options]), [LockEdits])***

Opens the record.

| Argument | Description |
|---|---|
| *`Source`* | The source of the recordset: a table name, query name, or SQL statement. |
| *`Type`* | The type of recordset to open: `dbOpenTable`, `dbOpenDynamic`, `dbOpenDynaset`, `dbOpenSnapshot`, or `dbOpenForwardOnly`. |
| *`Options`* | A combination of constants that specify characteristics of the recordset. See DAO Help for more information about these options. |
| *`LockEdits`* | The locking used by the recordset: `dbReadOnly`, `dbPessimistic`, `dbOptimistic`, `dbOptimisticValue`, or `dbOptimisticBatch`. |

The following code example opens the Employees table in the Northwind Traders sample database as a recordset and displays its contents on the active sheet:

```
Dim strDbPath As String
Dim db As DAO.Database
Dim rs As DAO.Recordset
Dim qt As QueryTable

strDbPath = "C:\Program Files\Microsoft Office\" & _
    OFFICE11\SAMPLES\Northwind.mdb"

Set db = OpenDatabase(strDbPath)
Set rs = db.OpenRecordset("Employees")

Set qt = ActiveSheet.QueryTables.Add(Connection:=rs, _
    Destination:=ActiveSheet.Range("A3"))

ActiveSheet.Range("A1") = qt.Recordset.Name & " table:"

qt.Refresh
```

### 12.8.1. DAO.DbEngine Members

| | |
|---|---|
| BeginTrans | CommitTrans |
| **`CompactDatabase`** | CreateDatabase |
| CreateWorkspace | DefaultPassword |
| DefaultType | DefaultUser |
| Errors | Idle |
| IniPath | LoginTimeout |
| OpenConnection | **`OpenDatabase`** |
| Properties | RegisterDatabase |
| Rollback | SetOption |
| SystemDB | Version |
| Workspaces | |

***dbengine.CompactDatabase(olddb, newdb, [locale]), [options] , [password])***

---

Copies and compacts a database. The database must be closed.

| Argument | Description |
|---|---|

| | |
|---|---|
| *olddb* | The name and path of the existing database file. |
| *newdb* | The name and path of the compacted database file. |
| *locale* | An optional collating order used in creating the compacted database file. See DAO Help for more information about collating order settings. |
| *options* | A combination of constants that specify characteristics of the recordset. See DAO Help for more information about these options. |
| *password* | An optional password string. |

***dbengine.OpenDatabase(dbname, [options], [read-only], [connect])***

Copies and compacts a database. The database must be closed.

| Argument | Description |
|---|---|
| *dbname* | The name and path of the existing database file. |
| *options* | A combination of constants that specify characteristics of the recordset. See DAO Help for more information about these options. |
| *read-only* | Use True if you want to open the database for read-only access. |
| *connect* | A connection string. |

See the `Database.OpenRecordset` method for an example of using `OpenDatabase`. Note that you use the `OpenDatabase` method without explicitly specifying the `DbEngine` object.

PREV PREV NEXT NEXT

## 12.9. DAO.Document and DAO.Documents Members

| | |
|---|---|
| AllPermissions | **`Container`** |
| Count[1] | CreateProperty |
| DateCreated | LastUpdated |
| **`Name`** | Owner |
| Permissions | Properties |
| Refresh[1] | UserName |
| [1] Collection only | |

### *Document.Container*

Returns the name of the container to which the document belongs.

### *Document.Name*

Returns the name of the specified table, query, form, or report. The following code example displays the names of all the reports in the Northwind Traders sample database in the first column of the active worksheet:

```
Dim strDbPath As String
Dim db As DAO.Database
Dim docRpt As DAO.Document
Dim intIdx As Integer

strDbPath = "C:\Program Files\Microsoft Office\OFFICE11\SAMPLES\Northwind.mdb"

Set db = OpenDatabase(strDbPath)
intIdx = 0
With db.Containers!Reports
    For Each docRpt In .Documents
        ActiveSheet.Cells(intIdx + 1, 1) = .Documents(intIdx).Name
        intIdx = intIdx + 1
    Next docRpt


End With
```

PREV PREV NEXT NEXT

## 12.10. DAO.QueryDef and DAO.QueryDefs Members

| | |
|---|---|
| Append[1] | CacheSize |
| Cancel | Close |
| Connect | Count[1] |
| CreateProperty | DateCreated |
| Delete[1] | **`Execute`** |
| LastUpdated | **`MaxRecords`** |
| Name | ODBCTimeout |
| **`OpenRecordset`** | Prepare |
| | |

| | |
|---|---|
| RecordsAffected | Refresh[1] |
| ReturnsRecords | **SQL** |
| StillExecuting | Type |
| Updatable | |
| [1] Collection only | |

### *querydef.Execute([Options])*

Executes the specified action query.

| Argument | Description |
|---|---|
| *Options* | A combination of constants that specify characteristics of the recordset. See DAO Help for more information about these options. |

### *querydef.MaxRecords[= setting]*

For and ODBC data source, returns the maximum number of records to return from the query.

### *querydef.OpenRecordset([Type], [Options]), [LockEdits])*

Opens the record.

| Argument | Description |
|---|---|
| *Type* | The type of recordset to open: `dbOpenTable`, `dbOpenDynamic`, `dbOpenDynaset`, `dbOpenSnapshot`, or `dbOpenForwardOnly`. |
| *Options* | A combination of constants that specify characteristics of the recordset. See DAO Help for more information about these options. |
| *LockEdits* | The locking used by the recordset: `dbReadOnly`, `dbPessimistic`, `dbOptimistic`, `dbOptimisticValue`, or `dbOptimisticBatch`. |

The following code example displays the contents of the recordset produced by the Invoices query in the Northwind Traders sample database on the active sheet:

```
Dim strDbPath As String
```

```
Dim db As DAO.Database
Dim qry As DAO.QueryDef
Dim rs As DAO.Recordset
Dim qt As QueryTable

strDbPath = "C:\Program Files\Microsoft Office\OFFICE11\SAMPLES\Northwind.mdb"

Set db = OpenDatabase(strDbPath)
Set qry = db.QueryDefs("Invoices")
Set rs = qry.OpenRecordset

Set qt = ActiveSheet.QueryTables.Add(Connection:=rs, _
    Destination:=ActiveSheet.Range("A1"))

qt.Refresh
```

***querydef.SQL[= setting]***

---

Sets or returns the query's SQL string.

PREV PREV NEXT NEXT

## 12.11. DAO.Recordset and DAO.Recordsets Members

| | | |
|---|---|---|
| AbsolutePosition | **AddNew** | BatchCollisionCount |
| BatchCollisions | BatchSize | **BOF** |
| Bookmark | Bookmarkable | CacheSize |
| CacheStart | Cancel | CancelUpdate |
| Clone | Close | Connection |
| CopyQueryDef | Count[1] | DateCreated |
| Delete | Edit | EditMode |
| **EOF** | FillCache | Filter |
| FindFirst | FindLast | FindNext |
| FindPrevious | GetRows | Index |
| LastModified | LastUpdated | LockEdits |
| Move | **MoveFirst** | **MoveLast** |
| **MoveNext** | **MovePrevious** | Name |
| NextRecordset | NoMatch | OpenRecordset |
| PercentPosition | RecordCount | RecordStatus |

| | | |
|---|---|---|
| Refresh[1] | Requery | Restartable |
| Seek | Sort | StillExecuting |
| Transactions | Type | Updatable |
| Update | UpdateOptions | ValidationRule |
| ValidationText | | |

[1] Collection only

***recordset.AddNew***

Adds a new record to the recordset. The following code adds a new record to the Employees table in the Northwind Traders sample database using cell values on the current worksheet:

```
Dim strDbPath As String
Dim db As DAO.Database
Dim rs As DAO.Recordset

strDbPath = "C:\Program Files\Microsoft Office\OFFICE11\SAMPLES\Northwind.mdb"

Set db = OpenDatabase(strDbPath)
Set rs = db.OpenRecordset("Employees")

rs.AddNew
rs!LastName = ActiveSheet.Range("B4")
rs!FirstName = ActiveSheet.Range("C4")
rs.Update
```

***recordset.BOF[= setting]***

True if the current record position is before the first record in the recordset.

***recordset.EOF[= setting]***

True if the current record position is after the last record in the recordset. The following code uses the `EOF` property to test for the end of the recordset, adding names from the Employees table in the Northwind Traders sample database to the first column of the current worksheet:

```
recordset.MoveFirst Dim strDbPath As String
Dim db As DAO.Database
Dim rs As DAO.Recordset

strDbPath = "C:\Program Files\Microsoft Office\OFFICE11\SAMPLES\Northwind.mdb"
```

```
Set db = OpenDatabase(strDbPath)
Set rs = db.OpenRecordset("Employees")

rs.MoveFirst
intIdx = 1
Do Until rs.EOF
    strName = rs!FirstName & " " & rs!LastName
    ActiveSheet.Cells(intIdx, 1) = strName
    rs.MoveNext
    intIdx = intIdx + 1
Loop
```

### *recordset.MoveFirst*

Moves to the first record in the recordset.

### *recordset.MoveLast*

Moves to the last record in the recordset.

### *recordset.MoveNext*

Moves to the next record in the recordset. See the `EOF` code example for an example of using `MoveNext`.

### *recordset.MovePrevious*

Moves to the previous record in the recordset.

PREV NEXT